MERCi-MIsS: Should I turn off my servers?

Mar Callau-Zori[1] Luciana Arantes[2] Julien Sopena[2] Pierre Sens[2]

[1] IRISA, Université de Rennes 1 - France. {firstname.lastname}@irisa.fr
[2] Sorbonne Universités, UPMC Univ Paris 06, CNRS, Inria, LIP6 - France
{firstname.lastname}@lip6.fr

Abstract. In recent years, the electrical consumption of data centers has increased considerably leading to a rise in the expenditure bill and in greenhouse gas emissions. Several existing *on/off algorithms* reduce energy consumption in data centers or Clouds by turning off unused (idle) machines. However, the turning off/on of servers consumes a certain amount of energy and also induces the wear and tear of disks. Based on the data streaming paradigm which deals with large amount of data on-line, we present in this paper MERCi-MIsS, a proposal whose aim is to save energy in data centers and Clouds and tackle the above tradeoff problems without degrading, as much as possible, the quality of services of the system. MERCi-MIsS dynamically estimates the future workload based on the recent past workload, deciding if servers should then be turned either on or off. We have implemented MERCi-MIsS on top of Twitter Storm. Evaluation results from experiments using real traces from Grid'5000 confirm the effectiveness and efficiency of MERCi-MIsS algorithm to save energy and avoid disk damage while the quality of service is only slightly degraded.

1 Introduction

In a Cloud environment, the provider renders available a great number of resources for clients to perform their tasks. Cloud computing has been presented as a green approach in front of traditional data centers since their resources are shared by a huge number of users, optimizing, thus, the use of the resources.

Although Cloud computing seems the correct approach for saving energy, more effort must be made in order to design efficient Cloud data centers [1]. In the Cloud, clients and providers have different responsibilities: the client is responsible for his/her application while the provider is interested in adopting energy-aware and cost effective policies. Furthermore, providers' energy-aware solutions should deal with a large number of applications. Therefore, based on a global view of the system, providers have to apply energy saving techniques which will not interfere in aspects which are responsibility of the clients.

One well-known approach to reduce energy consumption, called *on/off algorithm*, consists in turning off unused (idle) machines [2, 3], since the power of idle machines is estimated between 25-60% of the peak power [4, 5]. However, such an algorithm entails some negative impacts. Firstly, the turning off/on of servers consumes energy. Hence, a server should stay off during a minimum time

period (called *critical time*) which compensates the energy in rebooting it when compared with the energy of keeping it idle [3, 6]. A second negative impact is that the reduction of the number of available resources can degrade the quality of service (QoS) engaged by the provider through the Service Level Agreement (SLA), i.e., an agreement between the provider and the client which sets up the QoS that the provider should guarantee. The non satisfaction of SLAs could result in penalization to the provider. In this paper, we consider that the violation of SLA leads to monetary charges to the provider, i.e., the latter must reimburse the client if some service does not satisfy the SLA requirements. Finally, booting affects disk lifetime, i.e., the probability of disk damage, and thus replacement, increases with the number of boots [2, 7–11] . Thus, an energy saving solution should take into account the costs of the wear and tear of disks.

Considering the above discussed points, this paper presents MERCi-MIsS [3], a streaming-based algorithm which dynamically decides the number of servers to turn on/off. MERCi-MIsS proposes an energy saving strategy taking into account energy cost and disk wear-and-tear cost. MERCi-MIsS exploits a streaming model which is able to process great volume of data and, thus, decides on the fly about the number of servers to turn on/off. It exploits global system information, in terms of the number of required working, idle, off, turning on, and turning off servers. It also dynamically estimates the minimum number of idle servers which the system must keep in order to provide energy saving while ensuring the execution of unexpected works. We have also extended the *critical time* in order to take into account the wear and tear related to disk ignitions.

Performance evaluation experiments were conducted over traces concerning the usage of French Grid'5000 platform (a scientific experiment-driven research environment: www.grid5000.fr). Results confirm that MERCi-MIsS outperforms some energy saving algorithms found in the literature. It also provides shorter average time delay for processing clients' works than these algorithms.

Roadmap. Firstly, in Sec. 2, we discuss the minimum time that a server must be off in order to save energy boot. In Sec. 3, we present MERCi-MIsS, how it predicts the workload, computes both the monetary cost of non-working servers and of disk wear-and-tear. Evaluation is presented in Sec. 4. Sec. 5 discusses some related work. Finally, Sec. 6 concludes and proposes some future work.

2 Minimal period of time for off servers

The turning off and on of servers induces energy consumption. If we decide to turn off a server, it must be off for at least a minimum period of time which compensates the energy spent in rebooting it. In [3], the authors denote such a period of time the *critical time* (T_S). They also propose how to evaluate it.

Considering the parameters given in Table 1, the critical time T_S is the minimum period of time that a server is turning off which renders the energy spent in booting a server equals to the energy in keeping it idle, i.e., T_S such

[3] Maximizing Energy and disk ReplaCement saving — MInimizing SLA penalties

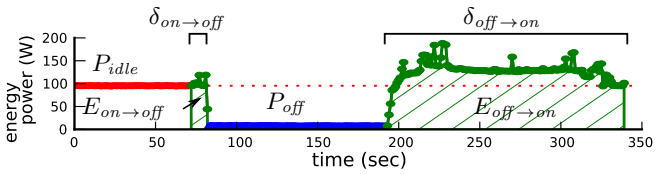


Fig. 1: Energy consumption of different states. Real experiment in Grid'5000

that $E_{idle}(T_S) = E_{reboot}(T_S)$, where the energy spent in T_S seconds of an idle server is $E_{idle}(T_S) = P_{idle} \times T_S$ while the energy for rebooting the server for the same period is $E_{reboot}(T_S) = E_{on\to off} + P_{off} \times (T_S - \delta_{tot}) + E_{off\to on}$, (the energy spent for both turning off and on the server plus the energy to keep it off). Hence, the critical time T_S is $\frac{E_{on\to off}+E_{off\to on}-P_{off}\times\delta_{tot}}{P_{idle}-P_{off}}$.

For instance, Fig. 1 shows an energy experiment conducted on a Dell Power Edge R720 server. The energy spent in turning on and off the server (green area in Fig. 1) is $E_{off\to on} + E_{on\to off} = 19,749$J, which respectively takes $\delta_{off\to on} + \delta_{on\to off} = 158$ seconds. Considering the average power of an off server and idle is $P_{off} = 8W$ and $P_{idle} = 97W$ resp., a power off server consumes in T_S seconds $E_{reboot}(T_S) = 19,749 + 8(T_S - 158)$ and an idle one consumes $E_{idle}(T_S) = 97T_S$. Hence, if the server keeps off at least $T_S = 208$sec., the decision of turning it off is an efficient one; otherwise, it is not worthwhile turning it off.

In the same paper, the authors argue that a T_S must be increased with the T_r factor which is related to the wear and tear with regard to the disk ignitions. However, they do not explain how to compute T_r.

We propose, therefore, in this article, an estimation for T_r. To this end, we add to $E_{reboot}(T_S)$ the energy cost (in Joules) associated with disk damage due to ignitions. Considering the cost of a new disk device (in money units) and the number of ignitions that a disk supports [8], the disk-cost of a boot (in money units) is estimated as $\$_B$. By dividing it by the cost of the energy $\$_E$, ($\$_B$ / $\$_E$), it is possible to estimate the energy spent in Joules due to disk damage. We have, then $E_{reboot}(t) = E_{on\to off} + P_{off} \times (t - \delta_{tot}) + E_{off\to on} + \$_B$ / $\$_E$. Thus, the *minimum critical time* T_S is:

$$T_S = \frac{E_{on\to off}+E_{off\to on}-P_{off}\times\delta_{tot}}{P_{idle}-P_{off}} + \frac{\$_B}{\$_E(P_{idle}-P_{off})} \quad (1)$$

In conclusion, an on/off algorithm must ensure this minimum critical time is used in order to both save energy and the cost of disk replacement.

$E_{on\to off}$	energy cost of turning off (J)	P_{idle}	energy power of an idle server (W)
$E_{off\to on}$	energy cost of turning on (J)	P_{off}	energy power of an off server (W)
$\delta_{on\to off}$	time spent in turning off (sec)	δ_{tot}	time spent in turning off&on (sec)
$\delta_{off\to on}$	time spent in turning on (sec)		$\delta_{tot} = \delta_{on\to off} + \delta_{off\to on}$
$\$_E$	cost of the energy ($ / J)	$\$_B$	cost of a boot ($)

Table 1: Event parameters for a single server

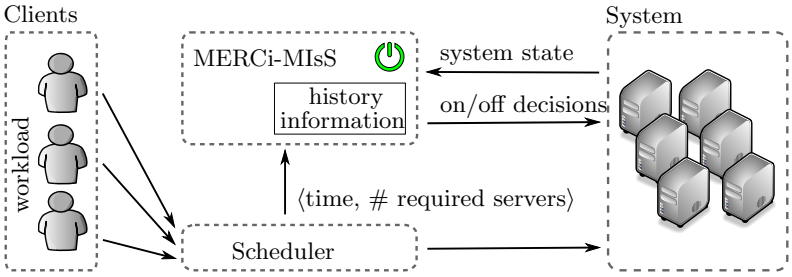


Fig. 2: MERCi-MIsS interaction with scheduler and system

3 MERCi-MIsS

MERCi-MIsS is an on/off algorithm based on streaming over sliding window model. That is, data are processed on-the-fly, continuously producing an output. We describe the MERCi-MIsS architecture in Section 3.1. On/Off algorithms turn on and off servers according to the needs of the system and prediction of future workload. Usually, algorithms estimate future workload based on previous one aiming at minimizing energy consumption as well as satisfying unexpected works, i.e., works that arrive when the system has not enough available servers. Thus, having a *minimum number of idle servers* helps to solve some unexpected situations. We denote m_0 such a minimum number of idle servers. In this case, at every time, the system can always process a new work that needs at most m_0 servers. Considering m_0 idle servers and the prediction of future workload based on the past workload, MERCi-MIsS decides about the number of servers to turn off or on at a given time. Section 3.2 describes how MERCi-MIsS takes decisions. In Section 3.3 we present how we evaluate the service maintainability cost associated with the energy spent in turn on/off servers and disk replacement.

3.1 MERCi-MIsS architecture

We consider that time is discretized in seconds, i.e., at every second it is possible to obtain the state of each server. At any time t, MERCi-MIsS needs the information about the current number of required servers and the current state of the system. While the former can be inferred from the workload with which the scheduler has to deal, the latter depends on the current processing works and might be affected by energy-aware policies. Figure 2 presents the architecture.

The number of required servers is predicted by MERCi-MIsS based on the history of clients' requests sent to the scheduler. Upon receiving a request, the scheduler decides when to execute the work. Notice, that, in some cases, clients must wait for their requests to be serviced (e.g., the system has not enough available servers). Hence, at any time t, the scheduler deduces the number of required servers to satisfy clients requests and providing the history of such a number to MERCi-MIsS, which stores it to predict future requirements.

Concerning the system state, MERCi-MIsS continuously receives information about it (the current number of working servers, idle servers, and off servers), producing as output the decisions about how many servers to turn off and on.

3.2 MERCi-MIsS turn on/off decisions

MERCi-MIsS exploits stream processing over sliding windows. As we have already discussed, the number of current required servers can be deduced by the scheduler workload. To this end, it keeps a window $\mathcal{W}$ with the most recent number of required servers, informed by the scheduler. Concerning the state of the system, at t, MERCi-MIsS receives as input $\langle n_w(t), n_{idle}(t), n_{off}(t)\rangle$ and produces as output the decisions about how many servers to turn off $d_{on\rightarrow off}(t)$ and how many servers to turn on $d_{off\rightarrow on}(t)$. Tab. 2 summarizes our notations.

One of the aims of MERCi-MIsS is to guarantee a minimum number, m_0, of idle servers at any time t. If some clients request more than m_0 servers, some servers must be turned on. On the other hand, when MERCi-MIsS decides to turn off some servers, it ensures that at least m_0 idle servers are on.

MERCi-MIsS, which decides either to turn on or off some servers, is described in Algorithm 1. We point out that both actions can not be taken at the same time since they are contradictory. If the system does not have a minimum of m_0 idle servers (lines 1-3), a number of servers will be turned on in order to ensure m_0 idle servers (at most we can turn on $n_{off}(t)$ servers). Otherwise, MERCi-MIsS tries to turn off some servers (lines 4-9), aiming at saving energy.

According to the critical time T_S, we can turn off all the servers which will not be used in the next T_S seconds (i.e. we need to estimate the maximum number of working servers in the next T_S seconds). However, the future workload is not known. Hence, MERCi-MIsS exploits the outliers border given in boxplot. The latter is a statistics graph where several descriptive values of a sample are represented. It shows five values from a data set: the upper and lower extremes, the upper and lower hinges (quartiles), and the median [12]. Values of the data set greater than the upper extreme are considered outliers. Hence, we can view the upper extreme UE, as a "normal" maximum bound of the data set. MERCi-MIsS estimates the future maximum number of working servers as the upper extreme value related to the number of working servers over the past history.

$n_w(t)$	nb. of working servers at t	$n_{on\rightarrow off}(t)$	nb. of servers turning off at t
$n_{idle}(t)$	nb. of idle servers at t	$n_{off\rightarrow on}(t)$	nb. of servers turning on at t
$n_{on}(t)$	nb. of power on servers at t $n_{on}(t) = n_w(t) + n_{idle}(t)$	$d_{on\rightarrow off}(t)$	decision about the number of servers to turn off at t
$n_{off}(t)$	nb. of power off servers at t	$d_{off\rightarrow on}(t)$	decision about the number of servers to turn on at t

Table 2: Servers type and decisions at time t

Algorithm 1: MERCi-MIsS algorithm

Parameters: m_0, minimum number of idle servers;
w, whisker length;
$\delta_{off \to on}$, the time spends in turn on
Input: $\langle n_w(t), n_{idle}(t), n_{off}(t) \rangle$, system state
Output: $d_{on \to off}(t)$, number of servers to turn off;
$d_{off \to on}(t)$, number of servers to turn on

1 **if** $n_{idle}(t) < m_0$ **then**
2 $\quad d_{on \to off}(t) = 0$
3 $\quad d_{off \to on}(t) = \min\{n_{off}(t), m_0 - n_{idle}(t)\}$
4 **else**
5 $\quad Q1 \leftarrow \texttt{quartile}(1, \mathcal{W})$
6 $\quad Q3 \leftarrow \texttt{quartile}(3, \mathcal{W})$
7 $\quad UE = Q3 + w\ (Q3 - Q1)$
8 $\quad d_{on \to off}(t) = \max\{0, \min\{n_{on}(t) - UE,$
$\qquad n_{idle}(t) + d_{off \to on}(t - \delta_{off \to on}) - m_0\}\}$
9 $\quad d_{off \to on}(t) = 0$

When the system has at least m_0 idle nodes MERCi-MIsS algorithm calculates the number of servers to turn off (lines 4-9). To energy efficiency, the number of servers to turn off is the number of servers not used within at least the next T_S seconds. In the current time t, the maximum number of servers to be used in the next T_S seconds is given by $n_m(t) = \max\{n_w(s) : s \in [t+1, t+T_S]\}$. Thus, we can turn off all the other servers which are on, i.e., $n_{on}(t) - n_m(t)$.

Note that we are considering that the number of future required servers, $n_m(t)$, is known at t and, in this case, $n_{on}(t) - n_m(t)$ represents the most efficient energy saving. However, this is not a realistic assumption since we can not foresee the future. Therefore, it is necessary to estimate $n_m(t)$ based on previous history of working servers. One first idea would be to use the maximum number of these servers in the recent history. Nevertheless, such an approach could induce a bad estimation if an unusual situation with high number of servers took place in recent history. In order to avoid such a mistake, MERCi-MIsS uses UE, the upper extreme value of boxplot, to estimate the number of working servers and the decision about the servers to turn off is (1) $d_{on \to off}(t) = n_{on}(t) - UE$. The upper extreme value UE is based on the first and third quartile[4] (respectively, $Q1$ and $Q3$) as well as a parameter w, called whisker length (usually $w = 1.5$). The upper extreme value is, thus, computed as $UE = Q3 + w\ (Q3 - Q1)$. Note that MERCi-MIsS computes quartiles over the sliding window $\mathcal{W}$ related to the number of required servers.

On the other hand, in order to ensure m_0 number of idle servers at time $t + 1$, the maximum number of servers to turn off at time t should be equal to

[4] Quartiles are ranked statistics which split data set into four equal groups. First quartile, Q1, is a value that is (equal or) greater that the 25% of the data values (resp. Q3 is equal or greater that the 75%).

$n_{idle}(t+1) - m_0$. However, since $n_{idle}(t+1)$ is unknown, MERCi-MIsS estimates the number of idle servers at time $t+1$ as the number of current idle servers plus the number of servers that MERCi-MIsS decides to turn on at time $t - \delta_{off \to on}$, i.e., such servers will be on at time $t+1$. Hence, for guaranteeing m_0 idle servers at $t+1$, we have that the number of servers to turn off satisfies (2) $d_{on \to off}(t) = n_{idle} + d_{off \to on}(t - \delta_{off \to on}) - m_0$.

Taking into account both conditions, i.e., the number of servers not used within at least the next T_S seconds and m_0 idle servers at time $t+1$, the number of nodes to turn off at t is equal to the minimum of (1) and (2) (line 8).

Exploiting system information. In the estimation of m_0 at $t+1$, MERCi-MIsS considers that the number of working servers at time $t+1$ is the same as the current number of working servers at t. However, there exist some cases where the system could give more information about the number of working servers and MERCi-MIsS could exploit it. For instance, if the workload was stored in a queue that MERCi-MIsS could have access to, the number of working servers at time $t+1$ could be inferred (provided that the workload queue is not empty).

3.3 Service maintainability cost

Service cost is composed of two costs: the *service performance cost*, associated with the clients' works execution, and the *service maintainability cost* related to the energy spent in turning on/off servers as well as disk replacement. One of the main goal of on/off algorithms is to reduce service maintainability cost as much as possible without degrading the QoS for the clients.

Service performance cost is related to the energy consumed by working servers. It is well-known that the energy spent by working servers depends on the work that must be executed, i.e., the clients' requests [13]. Estimating this energy consumption is not a trivial task. However, we can consider that a server which executes a given work spends the same energy regardless when the work is executed. In other words, the energy consumed by working servers to process a fixed workload is the same independently on the work that each server performs. Consequently, the service performance cost does not depend on the energy-aware policy. However, the turning on and off of servers introduces different energy consumption and disk replacements. The cost associated with them depends on the energy-aware policy and is considered as service maintainability cost. In this section, we focus in describe the service maintainability cost.

Service maintainability cost, $maintenance_{\$}$, has two parts: 1) $energy_{\neg w}$, monetary cost of energy of non-working servers (idle servers, off servers, and turning on and off actions); and 2) the monetary cost to replace disks.

$$maintenance_{\$} = \$_E \times energy_{\neg w} + \$_{replacement_{disk}} \quad (2)$$

At time t, the system has $n_w(t)$ working servers, $n_{idle}(t)$ servers, (i.e., $n_{on}(t) = n_w(t) + n_{idle}(t)$), $n_{off}(t)$ off servers, turning off servers ($d_{on \to off}(t)$), and turning on servers ($d_{off \to on}(t)$). Note that even if these values are related to time t, the evaluation of energy consumption concerns the whole period of time during

which the system is running. In the cost of $energy_{\neg w}$, both idle and off states are quite stable in terms of energy consumption. It is then possible to have representative average consumption values: P_{idle} and P_{off} power (Joules / sec) for idle and off servers respectively while the energy cost to turn on (respectively, off) a server is $E_{on \to off}$ (respectively, $E_{off \to on}$). Based on the energy parameters of Table 1 and the notations of Table 2, the energy consumed by non working servers, $energy_{\neg w}$, for the whole execution period of the system is given by:

$$\begin{aligned} energy_{\neg w} = & \sum_t \big(P_{idle} \times n_{idle}(t) + P_{off} \times n_{off}(t) + \\ & + E_{on \to off} \times d_{on \to off}(t) + E_{off \to on} \times d_{off \to on}(t) \big) \end{aligned} \tag{3}$$

The money cost associated with disk damage has a direct relation with the number of boots. As a boot is a turning off which will be eventually followed by a turning on, we cannot consider $n_{on \to off}(\cdot) + n_{off \to on}(\cdot)$ as the number of boots, otherwise, in the whole execution of the system, we would sum twice the number of boots. As a consequence, we consider the number of boots as the number of turning off $n_{on \to off}(\cdot)$ (eventually turning off servers will be power on). Hence, the disk money cost (in \$) is given by Equation 4.

$$\$_{replacement_{disk}} = \sum_t \$_B \times n_{on \to off}(t) \tag{4}$$

In Sec. 2, we defined T_S as the minimal critical time for saving energy which also includes the energy associated with disk replacement. Therefore, if T_S is respected, $maintenance_{\$}$ represents the minimum service maintainability cost.

Besides the monetary cost, $maintenance_{\$}$, we must consider the time delay to attend clients' requests which affects the quality of service. We propose a tradeoff metric based on the Energy-Delay product (EDP) [14], where the energy-performance tradeoff is evaluated by multiplying the energy by the time delay. To capture the disk damage we propose Energy&Disk-Delay product (EDDP) in Eq. 5 as the product of the energy consumed in the whole experiment (energy of non working servers plus disk replacement) by the average time delay to attend to clients' requests. Minimizing EDDP is equivalent to maximizing its inverse which represents the "performance-per-cost", where performance is the inverse of average time delay (service has low performance, if the time delay is high).

$$EDDP = \left(energy_{\neg w} + \frac{\$_{remplacement_{disk}}}{\$_E} \right) \times time_{delay} \tag{5}$$

$maintenance_{\$}$ estimation and EDDP concern all servers in the system during the whole experiment. However, considering just one server, we know that if it stays off at least T_S seconds, some energy is saved when compared to keeping it idle. In fact, the longer the period of time the server is off, the higher the energy saved. Hence, if a server keeps off Δt time, the service maintainability saved cost, denoted $saving_{\$}$, is given by Equation 6, where $E_{tot} = E_{on \to off} + E_{off \to on}$.

$$saving_{\$}(\Delta t) = \$_E \times \big((P_{idle} - P_{off}) \times \Delta t - E_{tot} + P_{off} \times \delta_{tot} \big) - \$_B \tag{6}$$

The minimum $saving_{\$}$ takes place at $T_S + 1$, i.e., $saving_{\$}(T_S + 1) = \$_E \times (P_{idle} - P_{off})$. Notice that, if a server is off $T_S + a$, the service maintainability saving is $saving_{\$}(T_S + a) = a \times \$_E \times (P_{idle} - P_{off})$.

4 Evaluation

In this section we firstly present the evaluation environment and input traces. Then we give a brief description of some algorithms with which we compared MERCi-MIsS, and finally, some comparative evaluation results are presented.

4.1 Evaluation Setup

MERCi-MIsS input (i.e., number of working servers) can be obtained by monitoring the states of the nodes or by inferring from users' reservation traces. We used real traces from [15] corresponding to 6 months (from 1st Feb. 2009 to 27 February 2010) related to reservations in Grid'5000 (12,948 reservations). Users made resource reservations indicating the submission time, the number of requested nodes, and the maximum duration of the reservations (however, users can cancel reservations before the ending time). Using the number of requested servers, the starting time, and the ending time, the number of working servers can be inferred. Assuming that the number of servers reserved by the users is the number of working servers, although users cannot use some of them, we assume that all the reserved servers must be on. Unfortunately, in the original traces, the actual ending time is not provided. Hence, we simulate this value considering the maximum duration as the actual duration. Energy values, cost, and duration are summarizing in Tab. 3. $E_{off \to on}$, $E_{on \to off}$, $\delta_{off \to on}$, $\delta_{on \to off}$, P_{off}, and P_{idle} are obtained from a real experiment where 20 Grid'5000 servers of the Lyon site, which represent more than 20% of servers of the site, were booted 50 times (the Lyon site has electrical consuming monitoring). The obtained results are similar to the ones presented in [3]. The costs of a boot $B_\$$ and the cost of energy $E_\$$ are taken from [8]. According to Sec. 2, the critical time T_S=1457 sec.

MERCi-MIsS evaluation experiments were conducted using Petrel-Storm on Grid'5000 platform. Storm [16] is an event processor to streams and Petrel-Storm is a tool for writing, submitting, debugging, and monitoring Storm topologies in Python [17]. By exploiting Grid'5000 traces, the input stream $S = \{R_t\}_t$ corresponds to a set of reservations R at time t. In the simulation, the interaction with the system which provides information about the system state (Sec. 3.1) does not take place. Instead, Storm operator maintains itself the system state $(n_w(t), n_{idle}(t), n_{off}(t))$. Hence, for each time t, the operator produces the decision about turning on $d_{off \to on}(t)$ or off $d_{on \to off}(t)$. Using the stream approach, we have implemented:

Perfect prediction. An ideal on/off algorithm which always has enough available servers and ensures the minimum $maintenance_\$$ cost. Thus, every ar-

$E_{off \to on}$	24,536.04J	$E_{on \to off}$	1,501J	P_{off}	9.58W	P_{idle}	150.16W
$\delta_{off \to on}$	120sec	$\delta_{on \to off}$	10sec	$\$_B$	0.5 cents/boot	$\$_E$	10 cents/KWH

Table 3: MERCi model parameters

riving work immediately starts executing without any delay. However, the perfect prediction is only feasible provided the future workload is known.

Turn-Off algorithm. In this algorithm, idle servers are always turned off. However, the algorithm does not ensure that a server stays off T_S seconds. Furthermore, the average time delay to satisfy clients' requests can be greater compared to other algorithms since the probability of having unexpected works which can not be immediately executed is higher than in an algorithm which always keeps some idle available servers.

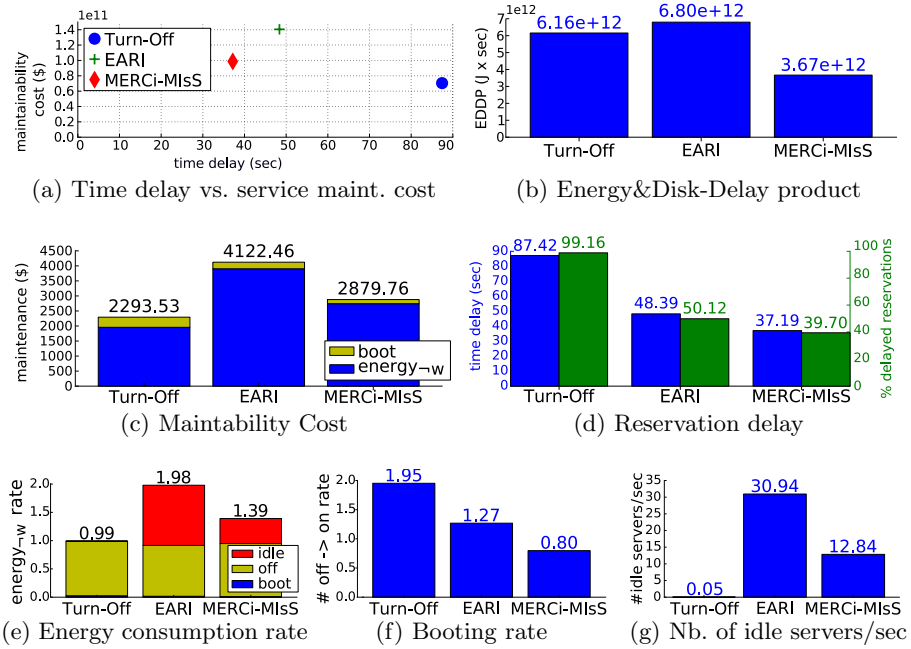
EARI [3]. An on/off algorithm for reservation-based systems (users reserve resources for a fixed time). EARI relies on the prediction of the next reservations. It estimates the number of servers to turn off whenever there are no waiting reservation requests to be scheduled. Nevertheless, no policy about turning on servers is described. Given M possible servers to turn off, EARI estimates the next reservation R with arrival time t using n servers. If R arrives before T_S seconds, then n servers stay on during T_S and $M - n$ servers are turned off. If after T_S seconds no reservation arrives, the above n servers are released, i.e., they will be considered to belong to the pool of possible servers to turn off. The estimation of reservation values (starting time t and number of servers n) is based on the history of previous reservations. Basically, the predicted value is the mean of the previous values ($mean(N)$) corrected with the mean of the previous errors ($mean(\mathcal{E}_N)$). Basically, the predicted value is the mean of the 5 previous values corrected with the mean of the 3 previous errors.

MERCi-MIsS. For performance evaluation, we consider a time-based sliding window of size 5min, slide of 3min, and the whisker length $w = 1.5$. While MERCi-MIsS bases its estimation on recent time (the last 5 minutes), EARI uses the last (5) reservation values. Notice that we could consider a longer time interval (till 3h) in EARI which would correspond to a much higher number of reservations. However, the risk of loosing the correlation between time and the number of reservations could greatly increase.

4.2 Evaluation results

In this section, we present a comparative by evaluating: 1) the tradeoff between the service maintainability cost and the average time delay to attend clients's requests; 2) the service maintainability cost; 3) the impact on the time delay and the number of delayed reservations; and 4) the processing time to take decisions.

Tradeoff between maintainability cost and time delay. Energy-aware policies must try to reduce service maintainability cost without increasing time delay for processing clients' work which degrades QoS. Fig. 3a shows the average time delay versus the service maintainability cost. The closer to the point (0,0), the lower the time delay and the service maintainability cost (better energy-aware policy). EARI has higher service maintainability cost and time delay than MERCi-MIsS. MERCi-MIsS has also a lower time delay than the Turn-Off policy. However, MERCi-MIsS has a slightly higher service maintainability cost than the latter. The results of Fig. 3b also confirm that MERCi-MIsS presents the



(a) Time delay vs. service maint. cost (b) Energy&Disk-Delay product

(c) Maintability Cost (d) Reservation delay

(e) Energy consumption rate (f) Booting rate (g) Nb. of idle servers/sec

Fig. 3: Turn Off, EARI and MERCi-MIsS performace

smallest EDDP (see Sec. 3.3). From both results, MERCi-MIsS has the best tradeoff between energy of non working servers, disk replacement, and time delay.

Service maintainability cost. Fig. 3c shows such a cost ($) for each algorithm. Blue and green portions of the bars are, resp., the cost related to the energy spent by non-working servers $energy_{\neg w}$ and disk replacement. Turn-Off is the best for monetary cost, but, it degrades the time delay as discussed later. In order to understand more deeply the service maintainability cost, we show different aspects: (1) Fig. 3e concerns the energy consumption of non working servers; (2) Fig. 3f is related to the number of boots (disk damage); and, (3) Fig. 3g shows the average number of idle servers per second. The energy bars in Fig. 3e are the energy consumed in the service maintainability divided by the energy in the perfect prediction algorithm. Three different colors make distinguishable the fraction of energy spent in different states (boot, off, and idle). As expected, Turn-Off algorithm consumes less energy in idle servers (the number of idle servers is close to 0 in Figure 3g). It is, thus, the best algorithm for saving energy. Notice that the number of idle servers in the perfect prediction is very low (0.62 server per second in average). On the other hand, MERCi-MIsS consumes 29% less energy than EARI. Fig. 3g confirms that such a reduction is due to idle servers. Bars in Fig. 3f represent the number of boots with regard to the perfect prediction. As expected, Turn-off performs the greatest number of boots which is almost twice the number in perfect prediction algorithm. In both Turn-

Turn-off	EARI	MERCi-MIsS
3.09	5.22	169.91

Table 4: Decision time processing in microseconds

off and EARI, the boot rate is higher than in the perfect prediction algorithm, contrarily to MERCi-MIsS, which presents lower boot rate than the latter (the ratio is smaller than 1). Fig. 3g, which shows the average number of idle servers per second, allows a better understanding of the different energy-aware policies. Turn-Off has a number of idle servers per second close to 0 (not 0 because a server must be in idle state to be turned off) while EARI has higher number of idle servers per second than MERCi-MIsS (2.4 times). Observing Fig. 3e-3g, we conclude that, during some periods, EARI maintains a large number of idle servers which are not required (EARI fails in the future workload prediction).

Reservation delay. Keeping servers in the off state has an impact on the QoS. Fig. 3d shows two results: 1) in the left side (blue), the average time delay for reservation; and 2) in the right side (green), the percent of delayed reservations. As expected, Turn-Off has a large number of delayed reservations (almost the whole reservation set) and the largest time delay. The impact of off servers on the QoS in the MERCi-MIsS is lower than in the EARI (shorter time delay and smaller number of delayed reservations). Therefore, in MERCi-MIsS, the number of off servers induces less degradation of the QoS than in the other algorithms. Such a result strengthens the previous one which concludes that MERCi-MIsS provides a better prediction of the future workload than the other algorithms. Comparing to the latter, it presents shorter time delay while using fewer resources. Hence, it has lower service maintainability cost.

Time for decision processing. On/Off algorithms should present a performance which allows the respective implementation in real environments. Tab. 4 summarizes the time spent to decide about the turn on/off actions. Obviously, the Turn-Off is the fastest one since no information is processed to take such a decision. EARI has a time processing close to Turn-Off due to the size of the processed information which is quite small (the last 5 reservations). MERCi-MIsS has the largest time processing because it considers the number of working servers of the last 5 minutes. However, we should emphasize that MERCi-MIsS time processing is feasible, i.e., 197 micro-sec while the time step is 1 sec. Hence, the three policies have time processing which are suitable for real environments.

5 Related work

In [18], authors present a survey on techniques for improving the energy efficiency of large-scale distributed systems. A taxonomy and survey of energy-efficient data centers and cloud computing systems can be found in [19].

The first on/off algorithm which considers disk damage was proposed in [2] where authors presented Muse, an operating system for a hosting center. The prediction approach focuses on estimating the resource demand of each customer considering her/his current request load level, contrarily to MERCi-MIsS algorithm, which characterizes the system load based on client demands, being, thus, more suitable for environments with a huge number of clients or with a dynamic set of users.

The concept of critical time, the minimum period of time which a server must be off to save energy, was introduced in [3]. The article then proposes the EARI algorithm for reservations-based environments such as Grid'5000, on top of which they conducted some evaluation experiments. We have extended their critical time concept with the time corresponding to the fraction that must be added to the former in order to consider the energy spent due to disk damage.

[20] presents two algorithms (online and offline) to turn off content delivery networks during periods of low load. The algorithms have three goals: maximize energy reduction, minimize the impact on client-perceived service availability, and reduce the wear-and-tear on hardware reliability. However, they have been designed to content delivery networks which operate as application service providers and can not be applied in other context such as infrastructure as a software (IaaS) or software as a service (SaaS).

The article [11] presents an online algorithm based on the number of active servers x_t at any time t. It uses a cost function to minimize some costs such as energy cost, cost related with network delay, and the cost of booting (including delay, mitigation, and disk damage). Nevertheless, as we have discussed in previous sections, the number of active working servers are not sufficient to compute the total energy cost because turning on and off servers consumes energy (analogous situation for power off servers). Therefore, an energy cost function must consider other server states than just active state. Concretely, the cost related to the disk damage is a linear function of the difference in consecutive times $x_t - x_{t-1}$. Hence, it is not fair to take into account just active servers such as in scenarios where, whenever one server concludes its turning on, the system decides to turn off one server. In this case, the number of active servers is always $x_t - x_{t-1} = 0$ and the model does not consider any disk damage.

A different approach of on/off algorithms is based on processor dynamic voltage/frequency scaling [8, 21]. However, processors consist of a small fraction of the total server power [22], entailing a moderate energy savings [13]. In [8], the authors consider disk damage to the dynamic voltage/frequency scaling strategy.

6 Conclusions and Future work

We have presented MERCi-MIsS whose aim is to reduce energy consumption in data centers without degrading the provided quality of services. MERCi-MIsS takes into account the energy spent by servers and disk damage due to wear-and-tear of ignitions and continuously decides how many servers to power off or on. We have conducted some simulation experiments based on real traces.

The results related to the Energy&Disk-Delay product (EDDP) metric, which expresses the above three aspects, confirm that MERCi-MIsS reduces in more than 39% the value of this metric when compared to the other algorithms.

As future work, we plan to evaluate heterogeneous systems by grouping servers according to their respective critical time and then applying MERCi-MIsS on each group. We will also evaluate the performance over other workloads.

Acknowledgements. This research is founding by the French National Agency for Research, ANR-10-SEGI-0009.

References

1. Cook: How clean is your cloud? Greenpace (2012)
2. Chase, Anderson, Thakar, Vahdat, Doyle: Managing energy and server resources in hosting centres. In: SOSP'01
3. Orgerie, Lefèvre, Gelas: Chasing gaps between bursts: Towards energy efficient large scale experimental grids. In: PDCAT'08
4. Fu, Lu, Wang: Robust control-theoretic thermal balancing for server clusters. In: IPDPS'10
5. Gulati, al.: VMware Distributed Resource Management: Design, Implementation, and Lessons Learned 2012
6. Benini, Bogliolo, Micheli: A survey of design techniques for system-level dynamic power management. VLSI'00
7. Elerath: Specifying reliability in the disk drive industry: No more MTBF's. In: RAMS'00. (2000)
8. Chen, Das, Qin, Sivasubramaniam, Wang, Gautam: Managing server energy and operational costs in hosting centers. In: SIGMETRICS'05
9. Bisson, Brandt, Long: A hybrid disk-aware spin-down algorithm with I/O subsystem support. In: IPCCC'07
10. Chen, Hu, Peng: Optimization of electricity and server maintenance costs in hybrid cooling data centers. In: CLOUD'13
11. Lin, Wierman, Andrew, Thereska: Dynamic right-sizing for power-proportional data centers. IEEE/ACM Trans. Netw. (2013)
12. McGill et al.: Variations of box plots. The American Statistician (1978)
13. Fan, Weber, Barroso: Power provisioning for a warehouse-sized computer. In: ISCA'07. (2007)
14. Gandhi, Gupta, Harchol-Balter, Kozuch: Optimality analysis of energyperformance tradeoff for server farm management. Perf. Eval. (2010)
15. `www.ens-lyon.fr/LIP/RESO/ict-energy-logs/index.html`
16. Storm. `storm-project.net` Petrel: `github.com/AirSage/Petrel`
17. `https://github.com/AirSage/Petrel`
18. Orgerie and al.: A survey on techniques for improving the energy efficiency of large-scale distributed systems. ACM Comput. Surv. (2013)
19. Beloglazov and al.: A taxonomy and survey of energy-efficient data centers and cloud computing systems. Advances in Computers (2011)
20. Mathew, Sitaraman, Shenoy: Energy-aware load balancing in content delivery networks. In: INFOCOM'12
21. Hotta, Sato, Kimura, Matsuoka, Boku, Takahashi: Profile-based optimization of power performance by using dynamic voltage scaling on a pc cluster. In: IPDPS'06
22. Meisner, Gold, Wenisch: Powernap: Eliminating server idle power. In: ASPLOS'09